"EXPRESS MAIL" MAILING
LABEL NUMBER:
EL448593683US

DATE OF DEPOSIT 12/10/99

I HEREBY CERTIFY THAT THIS
PAPER OR FEE IS BEING
DEPOSITED WITH THE UNITED
STATES POSTAL SERVICE
"EXPRESS MAIL POST OFFICE TO
ADDRESSEE" SERVICE UNDER 37
C.F.R. § 1.10 ON THE DATE
INDICATED ABOVE AND IS
ADDRESSED TO THE
COMMISSIONER OF PATENTS
AND TRADEMARKS,
WASHINGTON, D.C. 20231

_____
Debra J. Tix

"Middleware for Business Transactions"

By:

Edward Margoscin

Erica Quintana

Gerald A. Williamson

Debra K. Zarley

BACKGROUND OF THE INVENTION

1.   Field of the Invention

5   The present invention generally relates to computer software programs, and more particularly, the invention relates to an interface program between a business transaction server and a user interface.

2.   Description of the Related Art

10

Many companies, such as insurance companies and financial service organizations, have come to rely on software systems to help operate their businesses. The software systems may include a program or programs that process and store data associated with business transactions. Such programs may be referred to as business transaction servers. To process a particular business transaction, one or more business transaction servers may be used.

To use a business transaction server, an interface program may be needed to interact between the business transaction server and a user. Typically, the interface program receives input data from the user, transforms the data into a form that is recognizable to the business transaction server, and transmits the data to the server. The server receives the data, processes the data, and the server returns results to the interface program. The interface program receives the results from the server and the interface program may transform the results into a desired output form. The interface program may provide both a graphical user interface to a specific data input source, and a functional interface that communicates with a business transaction server.

There are many different systems and ways in which a user may input data. A system for inputting data into a computer system may be referred to as a "channel." Channels include, but are not limited to, office terminal entry systems, kiosk systems, internet systems, and telephone center assistance systems. A channel may be limited to certain types of transactions. For example, in the insurance industry, a kiosk entry system may be set up to process only insurance quotes, while an office terminal entry

system located in an insurance company office may be set up to allow all available types of insurance transactions.

The order in which data is requested from a user for a specific type of transaction may vary depending upon which type of channel is being used. For example, when a telephone assistance center entry system is used, one of the first pieces of information requested from the user is the user's name; while the user's name is often one of the last pieces information requested from a user when using an internet entry system. Because the order in which data is entered for different types of channels may vary, the interface program between a user and a business transaction server may be different for each different channel.
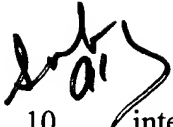
Typically, a company wants the graphical user interface that the company uses to be unique. Even if two companies use the same types of business transaction servers for the same types of business transactions, it is very likely that the graphic user interfaces for the two companies will be different. An interface program supplier may not be able to write one generic set of computer code that functions as an interface program for several different clients.

Typically, programmers write computer code to create a required interface program between a specific type of channel and a business transaction server or servers. If a company wants to offer the ability to enter data through a different type of channel that requires a different mode or layout for data entry, another interface program would have to be written for the new channel. Typically, programmers who create interface programs have to code both a graphical user interface and a functional interface to the business transaction server or servers. Writing computer code to provide the required interfaces typically requires specialist programmers who are familiar with the data requirements of the server, and who are also familiar with the specific requirements of data entry for the channel. Writing computer code to provide interfaces for two different types of channels may result in inefficient duplication of code in separate programs.

Developing an interface system between a user and a business transaction server could take months. One approach towards reducing development time is to use object-oriented software design techniques. Object-oriented software design may include the

use of objects and classes. An object may include an encapsulation of data and methods for manipulating the data. A class may include a template for an object. An object is therefore an instance of a class, and an object is created by instantiating it from a class. Object-oriented techniques, when used properly, may permit the re-use of previously

5   written program code such as classes.

SUMMARY OF THE INVENTION

10   A middleware program that provides a functional interface between a user interface and a business transaction server may in large part solve the problems outlined above. A middleware program is a software program that may function as an intermediary between an application program and a control-level program. The middleware program may serve as an intermediary between an interface to a channel and an interface to a business transaction server. The middleware program may allow a user

15   the ability to create a custom user interface for any desired type of channel without the need to create an interface to the business transaction server for each type of channel. The middleware program is expandable and changeable. The middleware program may be expanded or changed to reflect changes due to company policy change, due to regulatory change, or due to other factors.

20   The middleware program may be written in an object-oriented programming language. The middleware program may be easily extendable to include new objects. Preferably, extensions are created and placed in a customer created file or files. Alternatively, a customer could change the source code to extend the middleware program.

25   Previously, the development time for creating a functioning interface between a channel and a business transaction server could be months. The middleware program eliminates the need to create an interface between a channel and a business transaction server for each different type of channel. The middleware program may reduce the development time needed to provide a functioning interface system between a user and a

30   business transaction server. If a programmer uses the middleware program, a functioning interface system between a user and a business transaction server may be developed in a

few days or less, instead of months. Also, the specialization level of the programmers needed to develop the functional interface system is greatly reduced. Previously, programmers who understood the details of developing an interface between the user and a channel and who understood the details of developing an interface between the channel and the business transaction server were required to develop the interface system between the user and the business transaction server. With the use of the middleware program, programmers who develop the interface system need only have an understanding of the details needed to develop an interface between the user and the channel. The interface between the channel and the business transaction server is already developed and ready for use.

The middleware program functions as an interface between a channel and a business transaction server or servers. The middleware program may make the retrieval, display and updating of data easier to accomplish. The middleware program may allow a programmer to more easily develop a user interface for a computer software system that includes a business transaction server because the programmer does not have to develop computer code that interfaces to the business transaction server. The middleware program may include a variety of support processes. Such support processes may include, but are not limited to, support processes for data validation, support processes for performing syntax validation, and support processes for security checks.

The middleware program may save time, money, and reduce the degree of specialization needed by programmers who write computer code that provides an interface to the business transaction server or servers. The use of the middleware program may also eliminate the inefficient reuse of code in stand-alone programs that provide user interface systems for different types of channels. The use of the middleware program allows companies the opportunity to quickly and inexpensively create interface systems between users and business transaction servers. The user interface between a user and a channel may be customized to the specific technical requirements and aesthetic desires of each individual company.

Changing or modifying existing code of a software program so that the software program functions with a network system may be difficult, time consuming, and

prohibitively expensive. Such programs may be referred to as "legacy programs." The middleware program may allow the use of legacy programs on a network system without the need for modification of the legacy program. The middleware program may also allow the processing of business transactions that require access to multiple legacy programs running on different platforms.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is an illustration of a computer system;

Figure 2 is an illustration of a networked computer system;

Figure 3 is a block diagram of a business administration system that uses a middleware program;

Figure 4 is a block diagram of a portion of the hierarchical structure of classes for a middleware program;

Figure 5 is a partial listing of a text file of a domain source; and

Figure 6 is a flow diagram indicating the steps performed by the middleware program during the processing of a business transaction.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The term "computer system" as used herein generally describes the hardware and software components that in combination allow the execution of computer programs. Computer programs may be implemented in software, hardware, or a combination of software and hardware. Computer systems may take various forms, including, but not limited to, personal computer systems, mainframe computer systems, workstations, network appliances, internet appliances, personal digital assistants (PDA's), television systems or other devices. In general, the term "computer system" can be broadly defined to encompass any device having a processor that executes instructions from a memory medium. Figure 1 shows one embodiment of computer system 10. The computer system may include housing 12, a processor (not shown) that is located within the housing, memory medium such as floppy disks 14, display device such as monitor 16, and input devices such as keyboard 18 and pointing device (mouse) 20.

The term "memory" is used synonymously with "memory medium" herein. The term "memory medium" is intended to include an installation medium, e.g., a CD-ROM, or floppy disks, a volatile computer system memory such as DRAM, SRAM, EDO RAM, Rambus RAM, etc., or a non-volatile memory such as optical storage or a magnetic medium, e.g., a hard drive. The memory medium may comprise other types of memory as well, or combinations thereof. In addition, the memory medium may be located in a first computer in which the programs are executed, or the memory module may be located in a separate device or a second computer which connects to the first computer over a network. In the latter instance, the second computer provides program instructions to the first computer for execution.

The memory medium preferably stores a software program or programs. The software may be implemented in any of a variety of ways, including procedure-based techniques, component-based techniques, and/or object-oriented techniques, among others. For example, a software program may be implemented using ActiveX controls, C++ objects, JavaBeans, Microsoft Foundation Classes (MFC), or other technologies or methodologies, as desired. Software programs are written in a computer language or computer code. The code that forms a software program may be referred to as the source code of the program.

Computer system software generally includes at least one operating system, such as Windows NT, which is available from the Microsoft Corporation. An operating system is a specialized software program that manages and provides services to other software programs on a computer system. An operating system and the computer hardware that the

5   operating system run on may be referred to as "a platform."

Software may also include one or more programs to perform various tasks on the computer system or on various forms of data that are used by the operating system or other programs. The data may include but is not limited to databases, text files, and graphics files. Computer system software generally is stored in non-volatile memory or on an installation

10   medium. A software program may be copied into a volatile memory when running on the computer system. Data may be read into volatile memory as the data is required by a program.

One type of software program is referred to as a server. A server may be defined as a computer program that, when executed, provides services to other computer programs

15   executing in the same or other computer systems. A client program may be defined as a computer program that calls a server during execution. The computer system on which a server program is executing may be referred to as a server, and such a server may contain a number of server programs and client programs. In the client/server model, a server is a program that fulfills requests from client programs in the same or other computer

20   systems.

As used herein, the term "middleware" is a software program that may function as an intermediary between an application program and a control-level program. The application program may be an interface program between a user and a computer system, and the control-level program may be a server. The middleware program may be a client

25   program that calls a server. For illustration purposes only, the discussion of the middleware program for business transactions detailed herein will be described as being applied to a software system for the insurance industry. A server may process the business transactions. A server that may be used to process business transactions is the COGEN system. The COGEN system is supplied by Computer Sciences Corporation.

As used herein, the term "channel" generally describes any pathway that allows a user to interface with a computer system. Channels may include, but are not limited to, main office entry systems, branch office entry systems, kiosks, call centers, internet systems, and electronic data interchange systems. A channel may be a computer system. For

5   example, the computer system 10 shown in Figure 1 may be a channel used for an insurance office data entry system, a branch office entry system, or a component of the system used in a call center channel.

Each type of channel may require different implementation systems. For example, a kiosk entry system typically uses a touch-oriented visual screen. Special programming

10   implementations may be needed to transfer data from the kiosk to the computer system. A channel may be implemented so that the channel may only be used for selected types of business transactions. A kiosk system may be set up to only process certain business transactions, such as providing pricing quotes for specified insurance coverage situations. An office data entry system may use a monitor, keyboard and pointing device (mouse) to

15   allow for data entry into a computer system. A call center typically involves the oral communication of data to a person who enters the data into a computer system through a keyboard and/or pointing device. Electronic data entry systems may transfer data from one computer system to another.

A business transaction may be defined as a service provided by a company. For an

20   insurance provider, examples of business transactions include, but are not limited to, adding or deleting clients, tracking payables and receivables, establishing new policies, renewing policies, changing policies, canceling policies, providing pricing quotes for policies, and processing claims based on policies.

Figure 2 illustrates distributed or enterprise computing environment 22 according

25   to one embodiment. A distributed computer system or enterprise 22 may include a plurality of channels 24 that are interconnected through one or more networks. Although one particular embodiment is shown in Figure 2, the distributed computer system 22 may include a variety of heterogeneous channels and networks which are interconnected in a variety of ways and which run a variety of software applications and/or operating system

30   software.

One or more local area networks (LANs) 26 may be included in the enterprise 22. A LAN 26 is a network that spans a relatively small area. Typically, a LAN 26 is confined to a single building or group of buildings. Each node (i.e., channel or device) on a LAN 26 preferably has its own processor that may be used to execute programs.

5   Each node may be able to access data and devices anywhere on the LAN 26. The LAN 26 allows many users to share devices, such as printer 28, as well as data stored on file servers. The LAN 26 may be characterized by any of a variety of types of topology (i.e., the geometric arrangement of devices on the network), of protocols (i.e., the rules and encoding specifications for sending data, and whether the network uses a peer-to-peer or

10 client/server architecture), and of media (e.g., twisted-pair wire, coaxial cables, fiber optic cables, radio waves). As illustrated in Figure 2, the distributed computer system 22 may include one LAN 26. However, in alternate configurations the distributed computer system 22 may include a plurality of LANs 26 which are coupled to one another through a wide area network (WAN) 30. A WAN 30 is a network that spans a relatively large

15 geographical area.

One or more mainframe computer systems 32 may be coupled to the distributed computer system 22. As shown in Figure 2, the mainframe computer 32 may be coupled to the distributed computer system 22 through the WAN 30, but alternatively one or more mainframes may be coupled to the distributed computer system through one or more

20 LANs 26. As shown, the mainframe 32 may be coupled to a storage device or file server 34 and mainframe terminals 36. The mainframe terminals 36 may access data stored in the storage device or file server 34 that is coupled to or included in the mainframe computer system 32.

The distributed computer system 22 may include one or more channels 24 that

25 connect to the distributed computer system through the WAN 30. In other words, the enterprise 22 may include one or more channels 24 that are not coupled to the distributed computer system through a LAN 26. For example, the distributed computer system 22 may include channels that are geographically remote from the mainframe 32 and connect to the distributed computer system through the internet.

30 A business administration system is a computer system that may implement a series of instructions for processing business transactions. A business administration system

may include a plurality of channels and servers that may be networked together to form a distributed computing environment. A business administration system may be capable of performing multiple processing tasks. As used herein, a processing task may be defined as a sequence of one or more atomic transactions, or processing steps, associated with a

5   business transaction to be processed by the business administration system. As used herein, an atomic transaction may be defined as a sequence of information exchange and related work that is treated as a unit for the purposes of satisfying a business transaction request and for ensuring data integrity in the administration system. Examples of atomic transactions may include reading a value from a database, updating a field in a database,

10   and filling in a field of a form on a computer screen.

Figure 3 is a diagrammatic representation of an embodiment of insurance administration system 38 for a single channel 24. The insurance administration system 38 may include channel 24 and servers 40. The channel 24 may be any type of channel that provides an interface between a user (not shown) and the administration system 38. The

15   channel may include memory 14, input devices 42, and display 16. The memory 14 may store user interface program 44 and middleware program 46. Upon startup, the user interface program 44 allows a user to enter data and commands through the input devices 42. The input devices 42 may include a keyboard, a touch-screen, a pointing device such as a computer mouse, and/or any other device that allows data and commands to be transferred

20   to a computer system. As data and commands are entered with the input devices 42, the user interface 44 may display a representation of the data and commands on the display device 16. In an embodiment, the display device 16 may be a computer monitor. Data and commands that are entered by a user are transmitted from the user interface 44 to the middleware program 46. When the middleware program 46 receives sufficient data and a

25   command to initiate a business transaction, the middleware program transfers the data to a server 40. The use of more than one server 40 may be required to process a requested business transaction. The server or servers 40 process the data, perform the business transaction, and return the results to the middleware program 46. The middleware program 46 processes the results and returns the results to the user interface 44. The user interface

30   may then transmit the results to the display device 16.

The middleware program 46 may be a cross-platform implementation. A cross-platform implementation works on different computing platforms without alteration. Preferably, the programming interfaces are implemented in the Java™ Language for execution on a Java™ Platform. The Java™ Platform may provide a standard, uniform programming interface that allows Java™ applications to run on any computer system platform. The Java™ Platform is designed to provide this "write once, run anywhere" capability.

The Java™ Language is an object-oriented programming language. In an object-oriented programming language, data and related methods can be grouped together or encapsulated to form an entity known as an object. The object is the fundamental building block of object-oriented programming. The data structures within an object may alternately be referred to as the object's state, its attributes, its fields, or its variables. In the Java™ Language, the data structures are normally referred to as the variables of the object. The procedures that operate on the variables are referred to in Java™ as the methods of the object. The variables and methods of an object may all be referred to as the members of the object.

In object-oriented programming, the grouping together of the variables and methods within an object is referred to as encapsulation. When the variables relating to an object and the methods that might affect the object are encapsulated within the object, other entities usually do not have direct access to these data and procedures. The other entities instead call on the object itself to invoke its own methods and thereby operate on its own data. The encapsulation of the members of the object thereby provides some protection for the data within the object and prevents unauthorized, unwanted, or unintended manipulation of the data. This is sometimes referred to as data hiding. (The concept of data hiding through encapsulation should be distinguished from the hiding of variables in Java™ variable declarations, as explained in more detail below.)

If a user wants to hide the data within an object, the variable that contains the data is made private. Private variables within an object may only be accessed by the methods of the object. Because it may, in some cases, be inconvenient or impractical to require manipulation of certain data through the methods of the associated object, some variables

may be made public. These public variables are directly accessible to entities other than the object with which the variables are associated. Thus, in practice, the variables within objects normally comprise some that are hidden or inaccessible and some that are public.

All objects in an object-oriented programming system belong to a class, which can be thought of as a category of like objects which describes the characteristics of those objects. Each object is created as an instance of the class by a program. The objects may therefore be said to have been instantiated from the class. The class sets out variables and methods for objects that belong to that class. The definition of the class does not itself create any objects. The class may define initial values for its variables, and the class normally defines the methods associated with the class (i.e., includes the program code which is executed when a method is invoked.) The class may thereby provide all of the program code that will be used by objects in the class, hence maximizing re-use of code which is shared by objects in the class.

Classes in the Java™ Language may be hierarchical. That is, some classes may be subclasses of a higher class, also known as a superclass. In the Java™ Language, the subclass is said to extend the superclass. Alternatively, the superclass is said to be extended by the subclass. For the purposes of this disclosure, a subclass is considered to extend all or any of the classes that are above it in the hierarchy. It may also be said that the subclass directly extends the class immediately above it in the hierarchy, and indirectly extends higher classes in the hierarchy. For example, if a parent class is extended by a first subclass and that subclass is in turn extended by a second subclass, the second subclass can be said to extend the parent class as well as the first subclass.

The hierarchical definition of classes and subclasses based on shared variables and methods is very useful. A subclass includes all the variables and methods in the class of which it is a member (its parent class). The subclass is said to inherit the variables and methods of its parent class. This property is useful in defining subclasses because only those variables and methods that do not appear in the parent class need to be defined in the subclass (although variables or methods which appear in the parent class may be redefined in the subclass.) This allows the code written in the parent classes to be re-used so that the programmer does not have to rewrite or cut and paste code into each new

subclass. Methods that are defined in the parent class may, however, be redefined in subclasses. This is referred to as overriding or hiding the previously defined method(s). By redefining a variable that has already been defined in a superclass, the programmer may hide the previously defined variable (which is distinct from the object-oriented data-hiding concept inherent in encapsulation.) In some object-oriented languages, subclasses may inherit variables and methods from several classes. This is called multiple inheritance. If a subclass can only inherit from one parent class, this is called single inheritance. The Java™ Language is characterized by single inheritance, not multiple inheritance.

Hierarchical class structure also allows the programmer to take advantage of a property referred to as polymorphism. Polymorphism is a mechanism by which various objects may be handled in the same way externally, even though there are differences in the way they are handled internally. In other words, the interface that the different objects present to an external entity is the same for each object, but the details of each object's implementation may vary. This allows objects instantiated from different subclasses to be handled identically even though the subclasses are not identical. For example, assume that a drawing program implements a class for shapes, a subclass for circles, and a subclass for squares, each of which has a method called draw(). While draw() will be implemented differently for the circle subclass and the square subclass, the drawing program does not have to know the details of how a shape will be drawn, or even which of the shapes is to be drawn. The drawing program simply calls the draw() method for the object to be drawn and the implementation defined in the object's class will be used.

A variable within an object may be shared with another variable of the same type in a different object. When a variable is shared, the variable's value is derived from the current value of another variable, known as the source variable. The source variable must be available in the containment hierarchy of the object declaring the shared attribute. When the value of the shared variable is required, the middleware program 46 searches up the containment hierarchy to find the source variable. Having shared variables allows the middleware program to efficiently track changes to variables within the hierarchical

structure. The middleware program may have variables that track the status of other variables. A variable that tracks the status of another variable may denote if the tracked variable remains unchanged, if the variable is pending addition to a database, if the variable is pending deletion from the database, or if the variable has a new value that is to be updated within the database.

Figure 4 shows a possible subset of a hierarchical structure for the middleware program 46. Each class 48 includes variables 50 and methods 52. For each level in the hierarchy, the defined functionality is applied to that particular level and extends to all connected levels beneath it. The variables and methods defined in FSBusiness 54 apply to all the levels shown. The variables and methods defined in Policy 56 apply to PersonalAutoPolicy 58 and HomeOwnerPolicy 60, but the variables and methods defined in Policy do not apply to UnitAtRisk 62. Hierarchy of this type allows variables and methods to be re-used at the lowest level without having to redefine them. The heirarchy also prevents the user from calling methods that do not apply to the situation. For example, the hierarchy may prevent a user working with a home owner policy from attempting to perform a function that applies only to auto policies.

For the middleware program 46, a business object is an instantiation of a defined business class. The business classes define the available functionality that may be requested from the server. Within each class is the definition of all the individual fields or elements that are sent to or retrieved from a server 40, such as the COGEN system. Each class also has defined methods to set and retrieve these fields and to perform business functions on the objects as a whole.

To use the middleware program 46, a user enters data into the user interface 44 with the input devices 42. The user interface 44 transfers the data to the middleware program 46. The middleware program 46 keeps track of which variables have been changed. The middleware program 46 validates the data by checking if the entered value is within the domain of acceptable data. If the entered value is not within the domain, the middleware program sends an error code to the interface program 44. The interface program 44 would have computer code that allows entry of another value for the improperly entered value. The

error in the entry of a value may be noticed and corrected before the erroneous value is sent to the business transaction server 40.

A domain defines the valid characteristics for a business field, including, when appropriate, a list of the permissible values that the business field may contain. For example, valid values for marital status might be M = married, S = single, D = divorced, P = separated, and C = common law marriage. If a user mistakenly tries to enter "E" as a value for marital status, an error code would be sent to the user interface program 44. The name of the domain and its associated business field may be stored in a domain-attribute file. A domain source file may define the characteristics and, if appropriate, a listing of the valid values for the domain. A domain manager may be used to serialize both the domain-attribute file and the domain source file for use during program execution.

The attribute-domain source file is a single large text file that consists of statements linking each domain with an associated business field. An example of an entry in the attribute-domain source file that may be used to define a domain for use with the COGEN system is:

csc.fs.pc.policy.pauto.Vehicle~absBrakesCode~AbsBrakesCode

This entry states that there is a domain AbsBrakesCode, and that this domain is associated with the absBrakesCode business attribute field on the Vehicle business object in the package csc.fs.pc.policy.pauto. When the attribute domain source file is serialized, the file becomes a single, large serialized file.

The domain source file is a single, continuous text file that contains the source definitions for all domains required by the system. Figure 5 shows a portion of the statements that may be contained within the domain source file. The statements of Figure 5 state that there is a domain named "AbsBrakesCode," that the current valid values for the domain are N, O, R, S, and W, and that the English language translations for the code values are as shown. When the domain source file is serialized, the file is broken into separate serialized files for each domain. There may be several hundred serialized domain files during execution.

When the user is ready to initiate a business transaction, the user activates a control on the user interface 44. Activating the control may involve pressing a particular button, clicking or touching a particular area of a display screen, or typing in a particular key sequence. When the user activates the control, the middleware program 46 checks to see if

5   all of the required data needed to perform the transaction is present. For example, policy number is a required piece of data for a business transaction that adds a new beneficiary to an existing insurance policy, and the business transaction cannot be processed if the policy number is not included. If all of the required data to initiate a requested insurance transaction is not present, the middleware program sends an error code to the user interface

10   program.

If all of the required data to perform the requested business transaction is present, the middleware program gets the data that will be sent to the server. The middleware program only sends data that is required for processing the specific business transaction. If necessary, the middleware program puts the data in a required order. For example, assume

15   there are three classes A, B and C, and assume that A contains B and B contains C. The object model dictates the order A-B-C, but the server requires the order C-A-B. A method would be present in the code of the middleware program to override the A-B-C order and substitute the desired order of C-A-B. The server determines ordering requirements and the requirements are coded within the middleware program 46.

20   After ordering, the middleware program 46 may transform the data to be sent to the server into a form that is required by the server. If necessary, the middleware program 46 may transform values of data into forms that are recognizable to the server 40. For example, a user may enter a value of "M" or "F" to indicate gender, but the server 40 may require that the variable representing gender be either a 0 or a 1. The middleware program 46 would

25   send the appropriate value for gender to the server 40. The process of putting the data into a form that is required by the server 40 is referred to as flattening the data. After the data is flattened, the data is transmitted to the server 40, and the server processes the business transaction.

The server 40 may return results of the business transaction to the middleware

30   program 46. The middleware program 46 may take the results from the server 40 and put

the results in a form that is recognizable by the user interface program 44. The process of transforming the results of the server 40 into a form recognizable by the user interface 44 may be referred to as inflating the data. The middleware program 46 transmits the inflated data to the user interface 44, and the user interface may display the results of the business

5   transaction.

Figure 6 depicts the steps that the middleware program 46 performs during the processing of a business transaction. The middleware program 46 receives data from a user interface 44 to a channel 24. The middleware program 46 validates the integrity of the data against the domain of the data. The middleware program 46 receives a command from the

10   interface 44 that indicates that the user wishes to process a business transaction. The middleware program 46 gathers the data to be sent to the server 40 that will process the transaction. The middleware program 46 checks to see if there are values for all of the required variables. If all of the required variables are present, the middleware program 46 orders the data, flattens the data and sends the data to the server. The server processes the

15   data and returns the results to the middleware program 46. The middleware program 46 inflates the results and returns the inflated results to the user interface 44. The user interface may display the results.

The middleware program may be expanded to reflect changes in business policy, changes in regulations, or other changes. The source code of the program may be changed,

20   but it is preferred that the source code of the middleware program remain unaltered. Additions to the middleware program may be coded and placed in packages. The packages extend the functionality of the middleware program. For example, assume a car insurance company desires to instigate a new policy wherein the insurance company grants premium discounts that are a function of the color of the vehicle, but the server that the insurance

25   company uses to perform insurance transactions does not keep track of vehicle color. The new policy may be implemented in a relatively easy manner. The company would have to modify, or have modified, the insurance transaction server so that the server tracks vehicle color and calculates car premiums partially based upon vehicle color. The user interfaces from channels used to access the insurance transaction server would have to be modified so

30   that a value for the vehicle color may be entered. The domain may be extended to reflect

the possible values for the new variable. Also, an extension to the middleware program would need to be written that places vehicle color in an appropriate class, validates that a proper value for vehicle color is entered, and transfers the variable representing car color to the insurance transaction server at an appropriate time. The extension may be stored in a package that runs with the middleware program and extends the functionality of the middleware program.

Various embodiments further include receiving or storing instructions and/or data implemented in accordance with the foregoing description upon a carrier medium. Suitable carrier media include storage media or memory media such as magnetic or optical media, e.g., disk or CD-ROM, as well as signals such as electrical, electromagnetic, or digital signals, conveyed via a communication medium such as networks and/or a wireless link.

Although the system and method of the present invention have been described in connection with several embodiments, the invention is not intended to be limited to the specific forms set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents as can be reasonably included within the spirit and scope of the invention as defined by the appended claims.